NIKE DevOps Assignment

Nike DevOps Engineering Technical Test

Ming Li

2021

Table of Contents

[1. Introduction 2](#_Toc85232292)

[2. Architecture 2](#_Toc85232293)

[2.1 Prerequisites 2](#_Toc85232294)

[2.2 Solution Overview 3](#_Toc85232295)

[2.3 Git Repository 3](#_Toc85232296)

[3. Technology Choices 3](#_Toc85232297)

[4. Technical Questions 4](#_Toc85232298)

[4.1 Question 1 4](#_Toc85232299)

[4.2 Question 2 4](#_Toc85232300)

[4.3 Question 3 4](#_Toc85232301)

# Introduction

This is the Nike DevOps Engineering Technical Test. The objective of the test is to create the following:

1. Creating network infrastructure (VPC, VPN, SG)
2. Simple Spark Job running in EMR cluster which loads the data from CSV to S3 & Hive (Download simple EMR job from web)
3. Simple Java API which reads and writes the data to DynamoDB (Download simple REST API from web)
4. Terraform/CloudFormation script to create this environment along with the deployment of applications
5. CI/CD deployment (Optional)
6. Provide detailed architecture & guidelines for the new infrastructure setup

# Architecture

This is the architecture diagram of the solution -

Graphical user interface

Description automatically generated

**Figure 1: Architecture Diagram**

## Prerequisites

The following are the prerequisites before running the code

1. Multi-account Architecture -

The code assumes a multi-account AWS Architecture. The best practice for a secure and compliant environment is to create a AWS Landing Zone with Shared Service Deployment, Shared Service Development/Test/Production accounts.

1. CI/CD Pipeline in Shared Service Deployment Account

The code assumes that CodePipeline, CodeCommit and CodeBuild and provisioned in a Shared Service Deployment. Please use the attached CloudFormation template <<<>>> to create the CI/CD pipeline in Shared Service account. Code will be committed in the Shared Service Deployment Account and infrastructure would be deployed in Shared Service Development/Test/Production accounts using cross-account role and CloudFormation templates.

1. Cross Account Role in Shared Service Development/Test/Production Account

The code assumes that a Cross Account Role is created in the Shared Service Development, Test and Production accounts that grants Shared Service Deployment account to create CloudFormation template that can provision infrastructure – Lambda, EMR, S3, DynamoDB etc. in the Dev/Test/Prod accounts.

Please use the attached CloudFormation templates <<<xxx>>> to deploy cross accounts roles in the Shared Service Dev/Test/Prod accounts.

## Solution Overview

The solution consists of the following steps

1. Code Check-in in CodeCommit

The developer first checks the code in CodeCommit repository hosted in a Shared Service Deployment Account.

1. CodePipeline Orchestration

The checking of code in CodeCommit triggers CodePipeline which orchestrates the build. CodePipeline will first read the code from CodeCommit and then trigger CodeBuild to deploy the infrastructure in Shared Service Dev/Test/Prod accounts

1. Assume Cross-Account Role

A cross-account role in Shared Service Dev/Test/Prod account has a trust relationship with the Shared Service Deployment account. This role provides necessary permissions to CodeBuild in the Shared Service Deployment account to carry out deployment in the Shared Service Dev/Test/Prod Account.

1. Upload Code Package to S3

The CodeBuild project then uploads the packaged CloudFormation template and the code in the S3 bucket in the Shared Service Dev/Prod/Test account

1. Deploy CloudFormation Stack

The CloudFormation stack then provisions the Lambda function that gets triggered every time a CSV file is uploaded in the S3 Bucket. The lambda function creates a transient EMR cluster than reads the CSV file, processes it and copies the content in a DynamoDB table. A Java API allows the end user to GET/POST contents from the DynanoDB table.

## Git Repository

A bundled/archived repository showing your commit history or a link to an accessible private repository with your work in (Github Bitbucket). You could fork this repo in git, but any VCS is fine. Git example for sending us a standalone bundle:

git bundle create <yourname>.bundle --all –branches

This is the GIT repository --

# Technology Choices

The following is the rationale for Technology stack -

1. Multi-account AWS Architecture

We have deployed code using a cross-account CI/CD pipeline. In this setup, we have segregated the deployment account from the Infrastructure Dev/Test/Prod account. This segregation ensures appropriate control over the accounts in the infrastructure is provisioned and isolates the deployment environment from Dev/Test/Prod environment. The customer only needs to give access to Deployment environment to the developers and the infrastructure gets provisioned with the help of Cross-account roles.

1. Transient EMR cluster

The EMR cluster gets spun up by a Lambda function only when the CSV file gets uploaded in the S3 bucket. This cluster gets terminated automatically after it processes the file. This is a cost-effective measure and provisions infrastructure as and when desired

# Technical Questions

## Question 1

How long did you spend on creating the environment? What would you add to your solution if you had more time? If you didn't spend much time on the coding test, then use this as an opportunity to explain what you would add.

Answer: I spent 16 hours on creating the environment.

In the current solution, uploading a CSV file in the S3 Bucket triggers a Lambda function that launches the EMR cluster that processes the file. If I had more time, I would have added an SQS queue that gets S3 Event notifications, and which further gets processed by a Lambda function. This is because S3 triggering Lambda function is less reliable than an SQS queue.

I also could have added some unit tests --<<>>

## Question 2

What was the most useful feature that was added to the latest version of your chosen CloudFormation script? Please include a snippet of code that shows how you've used it.

<<<>>

## Question 3

How would you track down a performance issue in production? Have you ever had to do this the above task with the terabyte of data?

To track performance issues in production I would have loved to implement the solution that utilizes Prometheus, an open-source systems monitoring and alerting tool, and Grafana, an open-source visualization and analytics tool, to provide an end-to-end monitoring system for EMR clusters as outlined in this AWS Blog –

<https://aws.amazon.com/blogs/big-data/monitor-and-optimize-analytic-workloads-on-amazon-emr-with-prometheus-and-grafana/>

I have also used Sentry to track general issues in production.